Orbix 3.0c

Release Notes

February 1999

Contents

Orbix 3.0c

Release Notes

February 1999

Orbix 3.0c

Release Notes

February 1999

Introduction

These release notes provide details of changes made in Orbix 3.0. This document is divided into five main sections, each corresponding to one of the components of Orbix 3.0:

- Orbix
- Orbix Code Generation Toolkit
- OrbixCOMet
- OrbixNames
- Orbix Wonderwall

Orbix

This section describes changes made in Orbix 3.0.

Development Environments

This section describes the compiler and operating system versions that Orbix 3.0 has been built and tested with. The following applies to both multi-threaded and single-threaded variants of Orbix 3.0.

Solaris

Orbix 3.0 has been built on Solaris 2.5.1 using the SPARC C++ compiler version 4.2.

Patch 105568-11 or higher (`libthread.so.1` patch) for Solaris 2.6 should be installed on your system. This patch has a dependency on patch 105210 (`libc.so.1` patch), that is patch 105210 must be installed.

Testing of Orbix 3.0 was carried out on Solaris 2.5.1 and 2.6 and SPARC compiler version 4.2.

The Orbix 3.0 libraries contain RTTI information.

Year 2000 compliance

If you are using either Solaris 2.5.1 or 2.6, the Solaris recommended patch cluster should be installed as well as the patches below. More information about the recommended patch cluster is available at:

```
http://online.sunsolve.sun.co.uk/pub-cgi/
uk/pubpatchpage.pl
```

Note: Due to a known issue on Solaris 2.5.1, multi-threaded Orbix applications will not work on this platform if the year 2000 patches are installed. This issue has been reported to Sun Microsystems. When this issue is resolved, multi-threaded applications will work on this platform.

Solaris 2.5.1 Y2K Patches

103948-02 SunOS 5.5.1: accounting patch

103566-40 OpenWindows 3.5.1: Xsun patch

104463-03 SunOS 5.5.1: /usr/bin/date patch

104490-05 SunOS 5.5.1: ufsdump and ufsrestore patch

104816-01 SunOS 5.5.1: usr/sbin/sar patch

104818-01 SunOS 5.5.1: /usr/bin/passwd patch

104820-01 SunOS 5.5.1: /usr/lib/saf/listen patch

104822-01 SunOS 5.5.1: usr/lib/libadm.so.1 and usr/lib/libadm.a patch

104824-01 SunOS 5.5.1: usr/vmsys/bin/initial patch

104854-02 SunOS 5.5.1: troff macro patch

104873-04 SunOS 5.5.1: /usr/bin/uustat and other uucp fixes

105016-01 SunOS 5.5.1: usr/lib/libkrb.a and usr/lib/libkrb.so.1 patch

105675-01 SunOS 5.5.1: /usr/sbin/auditreduce patch

105701-02 SunOS 5.5.1: sysidsys unzip patch

104918-01 OpenWindows 3.5.1: y2000 filemgr patch

104995-01 OpenWindows 3.5.1: imagetool patch

104093-07 OpenWindows 3.5.1: mailtool patch

104977-01 OpenWindows 3.5.1: perfmeter patch

Solaris 2.6 Y2K Patches

105210-18 libc & watchmalloc patch

105393-07 at and cron utility patch

105464-01 OpenWindows 3.6: multiple xterm fixes

105621-09 libbsm patch

105800-05 /usr/bin/admintool patch

106193-03 sysid unzip patch

106828-01 /usr/bin/date patch

HP-UX

Two versions of Orbix 3.0 are available for HP-UX 10.20. Orbix 3.0 for the Cfront compiler has been built and tested with compiler version A.10.36 with patch phss-13124 installed. Orbix 3.0 for the ANSI C++ compiler has been built and tested with compiler version A.01.18 and language support library version A.01.18.

Testing of Orbix 3.0 was carried out on HP-UX10.20 on 700 and 800 series machines. Orbix has been compiled with `+DAportable`.

Orbix must now be compiled with the `-ext` flag to enable support for the `long long` and `unsigned long long` data types. Failure to do this results in compilation errors.

Threading Support on HP-UX

In order to use Orbix on HP-UX, the operating system threading support should have been installed using is the 'install and core OS' for HP-UX 10.20 CD and the subset DCE programming environment.

Year 2000 compliance

The following patch clusters should be installed on HP:

10.20 Series 700

Y2K1020S700

10.20 Series 800

Y2K1020S800

NT

Orbix 3.0 has been built and tested on NT 4 with service pack 4 installed, using the VC compiler version 6 service pack 2.

Win-95 and Win-98

Orbix 3.0 has not been built or tested on these platforms.

Compatibility with Other IONA Products

This section outlines the new Orbix 3.0 configuration mechanism and compatibility with previous versions of Orbix.

Version of IIOP Supported

Orbix 3.0 uses IIOP 1.0 by default. IIOP 1.1 is also supported.

Code set negotiation is not supported.

Version of CORBA

Orbix 3.0 supports version 2.1 of the CORBA specification.

Storage of Configuration Variables

The Orbix 2.x configuration mechanism of storing configuration variables in the `Orbix.cfg` file has changed in this release. In order to have a common configuration mechanism across IONA products; Orbix 3.0 introduces new 'scoped' configuration variables. These take the following form:

```
<IONA Product>.<variablename> = "<value>";
```

The previously used `Orbix.cfg` file has been replaced with a file named `iona.cfg` located by default in the `<iona_installation>/config` directory. This root configuration file contains links to all other IONA configuration files. There is also a special `common.cfg` configuration file. This contains Orbix configuration values that are used by multiple IONA products.

For backwards compatibility, Orbix uses any existing `Orbix.cfg` files; however, you should move existing configuration files to this new scoped format. For a full list of Orbix configuration variables, refer to the *Orbix C++ Administrator's Guide*.

How Orbix Finds its Configuration

Orbix has a chain of configuration handlers that it looks in when asked for a configuration parameter, these are (in order):

```
[Environment Handler ("IT_Environment")] →
[ScopedConfigFile Handler("IT_ScopedConfigFile")] →
[OldConfigFileHandler ("IT_ConfigFile")]
```

The Environment handler exists to allow any configuration variables defined in your environment to supersede those defined in configuration files or other user-defined configuration handlers.

To find the root configuration file (`iona.cfg` by default), the `ScopedConfigFile` handler checks the following:

- The environment variable `IT_IONA_CONFIG_FILE`.

  The configuration file need not be called `iona.cfg`.
- The environment variable `IT_CONFIG_PATH` and append `iona.cfg`.
- Looks for `iona.cfg` in the same directory as the Orbix runtime libraries.
- On Windows NT, checks the NT registry to find where Orbix was installed and appends `config\iona.cfg` to it.
- Tries the default installation locations (`c:\iona` on Windows NT, or `/opt/iona` on UNIX systems).

The dumpconfig Utility:

A utility named `dumpconfig` is provided in the `\iona\contrib\` directory. When run, this reports what configuration variables Orbix is picking up, and exactly where it finds the configuration files. It also reports if there are any syntax errors in

your configuration files that would normally be ignored by Orbix itself. Use `dumpconfig` if you are in any doubt about how Orbix is being configured.

Changes to PlaceCVHandlerBefore() and PlaceCVHandlerAfter() Code

If you are upgrading from Orbix 2.x and use the Orbix calls `CORBA::Orbix::PlaceCVHandlerBefore()` or `CORBA::Orbix::PlaceCVHandlerAfter()`, you need to change the code to specify `IT_ScopedConfigFile` instead of the old `IT_ConfigFile` or `IT_Registry` handlers.

Inter-Orbix Version Interoperability

This section details compatibility with previous versions of Orbix.

Repository IDs, Interface Markers, and Object Keys

This release of Orbix includes fixes or support for the following:

- Support for `#pragma` prefix.
- Support for `#pragma` ID.
- Full backward compatibility of Orbix object keys, including cases that use `#pragma` directives. This means that the `#pragma` directive has no effect on the object key.
- Narrowing of Orbix object references from previous version of Orbix clients and foreign ORBs.
- Remote `_is_a` operations from OrbixWeb and foreign ORBs in general. This is the way that most other ORBs perform narrowing.

CORBA Compliance

Orbix 3.0 can compile code with `#pragma` directives and publish IORs that both previous versions of Orbix and foreign ORBs can understand because Orbix 3.0 can service `_is_a` calls with OMG type IDs as parameters. Foreign ORBs can therefore narrow Orbix3.0 IORs and pre-Orbix3.0 IORs successfully.

TypeCodes

TypeCode usage with Orbix has been extended to cover single-level recursive definitions.

TypeCodes have also been modified so that they use correct repository IDs and type names. This improves interoperability for the `CORBA::Any` and `CORBA::TypeCode` types.

The IDL legacy flag `-typeCode` can be used to generate pre-3.0 TypeCodes. The `-typeCode` flag facilitates interoperation between Orbix 3.0 and previous versions of Orbix.

Interface Repository (IFR)

This is a list of rules for interoperability between Orbix 3.0 IFR and pre-Orbix 3.0:

- The 3.0 IFR must not use a pre-3.0 repository directory. If upgrading to Orbix 3.0, the Interface Repository directory must be cleared, or a new directory should be created.
- The version of `putidl` must be the same as the IFR version. For example, 3.0 `putidl` with 3.0 IFR, 2.3 `putidl` with 2.3 IFR.
- Different versions of Orbix clients can read from other versions of Orbix IFRs. For example, 3.0 `readifr` with 2.3 IFR.

Smart Proxy Factories

Const Correctness

The const correctness of the CORBA headers provided with Orbix has been improved for this release. As a side effect of this, some signatures have changed from `char*` to `const char*`. In most cases this is not a problem and is transparent to users of Orbix.

However, existing programs that use smart proxy factories are affected. The signature of the base ProxyFactory `New()` function has changed. Because the author of a proxy factory is required to override this method in their derived class, it is essential that the signature of the overriding method *exactly* match that of the base virtual method. Otherwise, the derived function hides but does not override the base method.

Because smart proxies have methods of the form `New(char*)`, and the new base method class has `New(const char*)`, this means that the factory uses the base (default) handling. In effect the proxy factory is disabled.

In order to correct this problem, add `const` to the `New()` methods that are being overridden.

Constructor

The proxy Factory constructor must now be called with the `_IR` name of the class; for example, a SmartProxy constructor for the grid would be as follows:

```
SmartProxyFactoryClass : CORBA::ProxyFactory (grid_IR) {}
```

In the past you could have used `grid_IMPL` in place of `grid_IR`.

Dynamic Allocation of Arrays

To allocate an array dynamically, a conforming program must use the functions, which are defined, at the same scope as the array type. For array T, these functions are defined as:

```
//C++

T_slice* T_alloc ();

void T_free ( T_slice* );
```

Failure to use these functions can lead to undefined behaviour. Refer to the chapter "The CORBA IDL to C++ Mapping" in the *Orbix C++ Programmer's Guide* for a full description of these functions.

NamedValue and NVList

- `NVList` is now conforms to the CORBA 2.1 specification.
- Previously, when inserting items into a `NamedValue` the value component was deep copied depending on the argument mode. This is no longer the case, that is deep copying is not carried out.

CORBA Module Scoping

`TypeCode`, `Principal`, and `NamedValue` must now be explicitly scoped by `CORBA::` in the IDL code.

WinMode—the ITG Replacement in ITM

Writing GUI applications with Orbix presents certain problems to the programmer. A GUI application is typically event driven, and so is an Orbix application. Therefore, you must ensure that both types of events are dispatched to their appropriate handlers.

There are essentially two ways of doing this. First, each event loop can be executed in its own thread. However, this method introduces the usual problems of multithreaded applications—thread-safety. Often, simple applications do not warrant the introduction of this overhead.

The second method involves multiplexing the separate event loops. In the case of GUI Orbix applications the simplest way is to keep the GUI message loop and enable Orbix events to automatically trigger corresponding GUI messages. This has the advantage of allowing the application programmer to choose whatever kind of Windows message loop that they deem necessary.

Orbix WinMode enables the latter method of multiple event loop execution. Therefore, it allows easy integration into GUI applications built with frameworks such as MFC.

WinMode Availability

Orbix WinMode is only available with the multithreaded Orbix library (`ITM.LIB`).

Enabling WinMode

WINMODE.H must be included to access the new functions. An excerpt from this header file follows:

```
class OrbixWinMode {

      public:

            bool BeginWinMode(HWND app_wnd, UINT
                 new_events_msg, UINT op_complete_msg);

            static bool BeginWinMode();

            static void EndWinMode();

      protected:

            virtual void ProcessWindowsEvents() = 0;

};
```

The simplest way of enabling WinMode is to call the static member function `BeginWinMode()`. It is not necessary to instantiate the `OrbixWinMode` class. You can disable WinMode later by calling `EndWinMode()`. While WinMode is enabled in this way Orbix requires that a Windows message loop be executing to respond to Orbix events.

WinMode can also be enabled in a more involved way. This alternative way allows greater flexibility and improved integration into GUI application frameworks such as MFC. To enable WinMode in this way perform the following steps:

1. Inherit from `OrbixWinMode`.
2. Provide an implementation for `ProcessWindowsEvents()`.
3. Instantiate an object of your defined type.
4. Call the member function `BeginWinMode(HWND,UINT,UINT)`.
5. You can disable WinMode by calling `EndWinMode()` as before.

You must provide values to `BeginWinMode(HWND,UINT,UINT)`. The first parameter is a window handle that receives messages sent by Orbix to inform the application of relevant events. The next two parameters are message identifiers for those messages. The second parameter is a message Orbix sends to the appointed window when there are new Orbix events to be processed. The message handler for this message should call `CORBA::Orbix.processEvents(0)`. This dispatches all pending Orbix events.

The third parameter specified to `BeginWinMode(HWND,UINT,UINT)` is related to the `ProcessWindowsEvents()` virtual function. When Orbix makes a remote invocation it calls `ProcessWindowsEvents()`. This allows the application to process a Windows message loop that includes such features as accelerators or one that is implemented in a GUI framework such as MFC. When Orbix receives the reply for the operation, a message is sent to the appointed window to notify the application. This message identifier is specified in the third parameter to `BeginWinMode(HWND,UINT,UINT)`. When this message is handled the application should exit from the message loop and `ProcessWindowsEvents()` should be allowed to return.

IDL Compiler Errors

The following IDL extract contains an example of illegal IDL:

```
   module M {

1.  typedef long Long; //Long clashes with long

       typedef long TheThing;

       interface I {

          typedef long Mylong;

2.        mylong op1 (//miscapitalization of mylong

3.            TheThing thething //thething clashes with
TheThing

          );

      };

   };
```

This IDL is described as follows:

1. The compiler now emits an error of the form “Long clashes with keyword”.
2. A warning of the form “miscapitalization of mylong” is output.
3. The compiler will ouputs the warning “thething clashes with TheThing”.

Warnings 2 and 3 will be upgraded to error conditions in a future release.

It is now incorrect to forward declare an IDL interface without providing a proper declaration of that interface at some other point in the IDL specification being compiled. This is contrary to previous behavior, which was non-compliant. See the CORBA Specification version 2.1, Section 3.5.2

OBJECT_NIL

`OBJECT_NIL` is in the `CORBA` namespace and must be qualified when used. It is now properly type safe because it has the type `const CORBA::Object_ptr`. Before, this was essentially zero and was compatible with all pointer and arithmetic types (and those that accepted zero for construction).

Functionality Removed from Orbix 3.0

The following functionality has been removed from Orbix 3.0.

IR

The IR is no longer shipped with Orbix. This has been replaced by the IFR.

Static Bridge

The files for the static COM/CORBA bridge (`winidl` compiler, demos, and so on) are no longer shipped with Orbix. The OrbixCOMet product has replaced this functionality.

Orbix 3.0 Libraries

The Orbix 3.0 libraries are now as follows:

Static Libraries

Static libraries are no longer shipped with Orbix on any platform.

Unix Libraries

`liborbix` now contains `libITini` and `libDSI`.

`liborbixmt` now contains `libITinimt` and `libDSImt`.

Windows NT Libraries

The libraries shipped with Windows NT on Orbix include the following two libraries only:

`ITCi.lib` and `ITMi.lib`.

The following libraries are now part of the Orbix libraries:

`libDSICi.lib` `libDSIMi.lib` `ifr.lib` `initsvr.lib`

As mentioned previously, the static libraries are no longer shipped with Orbix. Thus the following libraries are no longer shipped:

`IRCLT.lib` `ITC.lib` `ITM.lib` `libDSI.lib`

The GUI tools and static bridge libraries are no longer shipped with Orbix. The following libraries have been removed:

`ITG.lib` `ITGi.lib` `ITOLEi.lib` `LibDSIGi.lib`

Refer to the section on WinMode on page 6 for details of the ITG replacement on NT.

New Features in Orbix 3.0

This section describes the new functionality and major changes added in Orbix 3.0.

Documentation

The Orbix user documentation has been updated for this release. New editions of the following manuals are shipped with the product:

- *Orbix C++ Programmer's Guide*
- *Orbix C++ Programmer's Reference*
- *Orbix C++ Administrator's Guide*

Orbix Demos

A number of the demos shipped with Orbix have been redesigned. The documentation has been updated to reflect the new demos.

Limiting Network Access Time

Orbix 3.0 has incorporated new features which gives the application more control over the low level behaviour of TCP network access.

In previous versions of the product, it was possible to assign a time limit to the duration of a full invocation including both the request and response. This was achieved by using the following APIs:

- `CORBA::ORB::defaultTxTimeout()`
- `CORBA::Environment::timeout()`

To ensure that attempted TCP connections did not overrun a set time limit the following APIs were used:

- `CORBA::ORB::abortSlowConnects()`
- `CORBA::ORB::setConnectionTimeout()`.

In Orbix 3.0 the application timeouts have been extended to cover the network access at a smaller granularity, thus ensuring that Orbix is sufficiently flexible and responsive when dealing with the demands of large and complex TCP networks under differing load characteristics. The API's mentioned above still exist, and operate in a similar manner to previously released versions of Orbix.

The default behaviour of Orbix 3.0 is the same as in previous versions of Orbix:

- Default `tx timeout` is `INFINITE_TIMEOUT`.
- Default `abortSlowConnects()` is `OFF`.
- Default `connectionTimeout` is 30 seconds.

The non-default behaviour is described below.

Aborting Slow Connections

When an Orbix client or server attempts to access a server, it goes through an initial TCP connection phase. The time spent in this activity can be quite large, particularly if the target server is extremely busy, or the target server's host is not on a route accessible from the local host. This can cause unacceptable delays in large network systems.

To allow the application to avoid such delays, it is possible to reduce the time the ORB will spend attempting to connect to a server by using the following API invocation:

```
CORBA::Orbix.abortSlowConnects(1)
```

After this call, any connection attempts that have not succeeded or failed within 30 seconds will return failure. By default, the ORB will try 10 times to connect to the target server, and each connection will apply this 30 second limit. Both the number of retries and the connection time limit can be altered by using the `CORBA::ORB::maxConnectRetries()` and `CORBA::ORB::setConnectionTimeout()` APIs respectively.

Previous versions of Orbix on UNIX platforms used the `SIGALRM` signal to effect the connection time limit. This feature has been changed in Orbix 3.0 on UNIX systems so that there is no use of signals.

Timing Out Slow Message Sends and Receives

In previous versions of Orbix, it was possible to cause a client to hang when the operating system's internal TCP message buffers became full. This situation could arise, for example, when the client was sending a large number of messages rapidly, in succession, and where the server processed the messages one or more orders of magnitude more slowly.

When a timeout for an invocation is set, either through the `CORBA::Orbix.defaultTxTimeout()` or `CORBA::Environment` APIs, that timeout now applies, separately, to both the sending and receiving part of the invocation. The result is that, if an invocation is given a timeout of 400 milliseconds and either the sending part or the receiving part of the invocation exceeds this amount of time, a CORBA system exception will be returned to the caller, stating that the operation has timed out.

Judicious use of these API's can permit an application to have more control over its runtime behaviour.

Cancelling of Bad Connections

It is possible that dysfunctional or malicious clients may attempt to connect to Orbix servers, especially the Orbix daemon, which usually has a well-known port number. To forestall the chance of destructive behaviour, Orbix 3.0 still cancels any unrecognised protocol access and adds a new feature—connection attempts to the daemon or any Orbix server which take longer than a default of 30 seconds are dropped by the daemon or server. This is the default behaviour. The value of this timeout can be changed using the `CORBA::ORB::setConnectionTimeout()` API.

Service Contexts

Service contexts are an IIOP-compliant way of implicitly passing service-specific information with IIOP requests and replies. The interoperability specification defines a mechanism for identifying and passing this service specific information as "hidden parameters".

Refer to the "Service Contexts" chapter in the *Orbix C++ Programmer's Guide* for a description of the Orbix APIs that provide the mechanism to supply and consume context information.

Orbix Daemon

The Orbix daemon is Java-enabled. This means the Orbix daemon can now launch Java servers.

New NT Daemon Flags

`-j` Install daemon as an NT service.

This starts the daemon with `<path>\orbixd -b`.

`-w:` Uninstall daemon as an NT service.

`-b:` Run daemon as an NT service.

New IDL Types

Support has been added for the following types: `long long`, `unsigned long long`, and `fixed`.

Note that there is a maximum allowed size for a fixed value. This is the maximum of a double value.

New IDL Compiler Flags

The following flags have been added to the IDL Compiler:

- `-Bonly`

  The same as `-B` flag, but also suppresses generation of TIE code.

- `-C`

  This flag has been removed. This means that comment filtering is no longer supported.

- `-typeCode`

  This flag is used with the `-A` flag and indicates that pre-Orbix 3.0 TypeCodes should be generated.

IIOP

A number of new configuration variables have been added for IIOP:

- `IT_IIOP_VERSION`

  This specifies the IIOP version of the IORs generated by Orbix servers, and of IIOP messages understood by Orbix. Valid values are 10 and 11, representing IIOP 1.0 and IIOP 1.1 respectively. The default value is 10.

- `IT_ONEWAY_RESPONSE_REQUIRED`

  This specifies if an IIOP reply is expected for an outgoing IIOP request containing a oneway operation. A response to a oneway is desirable when the user wishes to catch system exceptions, or to enable the client to receive IIOP replies with `LOCATION_FORWARD` status. However there is a performance overhead. Valid values for the variable are TRUE and FALSE, the default is FALSE. Oneways that require a response are not compatible with earlier versions of Orbix. Also, if the invocation is dynamic (made by the user using the Dynamic Invocation Interface), this variable is ignored.

Dynamic Invocation Interface

The following changes have been made to the Dynamic Invocation Interface (DII):

- In previous versions of Orbix the DII required the client application to be linked with the Client Stub code in order that User Defined Types code be marshaled and unmarshaled. This is no longer the case.
- Opaque data types cannot be used with the DII.
- The operation `CORBA::Request::assumeOrigArgsOwnership()` has been renamed `CORBA::Request::assumeArgsOwnership()`.

CORBA::Any

- In previous versions of Orbix the client stub code was required to be linked with the client application in order that user-defined types could be marshaled, unmarshaled, copied, and released. This is no longer the case.
- Orbix now provides a non-copying version of the extraction operator used for extracting an Any from within an Any:

```
// C++

Boolean operator >>=(Any *&) const;
```

New and Modified APIs

This section described new APIs added to Orbix 3.0 and existing APIs that have been modified.

Service Context APIs

The *Orbix C++ Programmer's Guide* explains the APIs provided for service contexts.

Proxy Server APIs

Two APIs have been added to `CORBA::Object` to provide support for Orbix Wonderwall:

- `CORBA::Object::enableProxyServer ( )`
- `CORBA::Object::setProxyServer ( )`

enableProxyServer ()

Synopsis

```
CORBA::Object::enableProxyServer (
    Boolean useProxy );
```

Description This API is provided for supporting Orbix Wonderwall.

If set to `true`, the object reference contains the host name and port number of the proxy server, that is Orbix Wonderwall. If set to `false` the object reference contains the actual server host and port.

This API may be called repeatedly for an object, and thus enables you to selectively publish object references with or without the proxy host name and port number.

Parameters

`useProxy`	`true` means the object reference contains the port and host of the proxy server. `false` means the object reference contains the actual server host and port.

Notes Orbix specific

setProxyServer ()

Synopsis

```
CORBA::Object::setProxyServer (
    const char * host, unsigned short port );
```

Description This API is provided for supporting Orbix Wonderwall.

It sets the values for the host name and port number for the proxy server (Orbix Wonderwall).

If `CORBA::Object::enableProxyServer ()` is set to `true`, the object reference will contain the host name and port number as specified by this API.

Parameters

`host`	The name of the host on which your firewall proxy server is running
`port`	The port number on which your firewall proxy server is listening.

Notes Orbix specific

I/O Streams

Two new defines have been added to the public API for Orbix: `IT_NOIOSTREAM` and `IT_USE_STD_IOSTREAM`.

`IT_NOIOSTREAM` switches off all reference to 'iostreams' in the Orbix external headers. It is most likely that this will be used by customers developing GUI applications.

`IT_USE_STD_IOSTREAM` is only meaningful when `IT_NOIOSTREAM` is not defined. It causes the external Orbix to use `<iostream>` instead of `<iostream.h>` and to use `std::cout` instead of `cout`, and so on.

This latter option is not present on platforms that do not have ISO standard streams available.

Multi-Homed Hosts

Multi-homed support for machines with multiple IP addresses is provided in Orbix 3.0. You can enable multi-homed support by defining the configuration variable:

```
Orbix.IT_ENABLE_MULTI_HOMED_SUPPORT = YES
```

You can also define the environment variable:

```
IT_ENABLE_MULTI_HOMED_SUPPORT = YES
```

It is disabled by default and will impact performance when enabled.

Check the IONA knowledge base for further information.

Deprecated Features

The following features are still supported, however their use is deprecated:

- `_bind()`

  You should now use OrbixNames.

- Transformers (for modifying marshaled data).
  You should now use Orbix SSL.

- Filters—using filters to piggy-back data.

  You should now use Service Contexts.

- Opaque data type.

- The Orbix network protocol.
- IDL compiler flags `-i` and `-f`.

Bugs Cleared in Orbix 3.0

This section describes the incidents cleared in this release. All incidents are cross platform unless otherwise stated. The incidents are broken down by module and described in terms of the following:

- **Incident ID**

  This is the reference number used by the development teams to track incidents, which may in turn relate to one or more PRs (problem reports) as reported by customers.

- **PR Number**

  Not all incidents fixed will have a PR number (the number assigned by IONA support when a call is logged).

- **Synopsis**

  This is a short description of the reported problem. A description of the fix is included in the fix where necessary.

IDL Compiler

Incident ID	PR Number	Synopsis
176		Type check not performed in const assignment for non-basic type.
215		The compiler does not report an error if an interface is referenced by a forward declaration but never defined.
370	14907	Generated code for sequences of types defined in modules does not compile.
	103316	
	171604	
385	13924	Invalid sequence type in generated code for typedef of data type defined in another module and contained in an included file.
	17170	
	17623	
	18107	
	21106	
	23708	
	142442	
402		A combination of `#include<orb.idl>` and `idl -N` generates code that will not compile.

559		Compiler does not check if scoped name resolves to a legal type in union `switch` statement.

587	13667	Problem with a series of sequence declarations such as the following:
		`typedef sequence<octet, 11> Octet11;`
		`typedef sequence<octet, 255> Octet255;`
		`typedef sequence<octet, 256> Octet256;`

605	115324	TypeCode in IDL should be scoped with `CORBA::`.
607		Defining an `enum` as `const` and assigning it causes a core dump.
610	144336	Comment treated as a reserved word in Orbix 2.3.
626		On Windows NT, the order of `CALL_SPEC` and `DECL_SPEC` for VC5.0 breaks VC4.2.
10883		Problem dealing with typedefed sequences of typedefed sequences.
11140	132437 133986 161655 164326 165740 167998	Arrays of typedef types produce generated code that does not compile.
11440	112195 133663 138364 133709 136980 190421	IDL compiler does not scope correctly within a module when using multiple inheritance.
12040	138367 121788	Incorrect server skeleton code generated by IDL compiler for opaque types.
12240	133779 148901	Issues with module reopening.

	150943	
	166849	
	187216	
	117565	
12840	119643	When using the `-N` switch on Windows NT, the Orbix 2.3 IDL compiler fails to compile a large IDL specification consisting of a `#include` directive within a module definition.
	125514	
	134854	
	161901	
12880	158526	The `-O` switch causes a segmentation fault.
	114309	
	119032	
	123818	
	125172	
	130927	
	147779	
	153558	
	154950	
	160639	
13381	134250	Issues with module reopening.
	136054	
	136918	
	149057	
	154345	
	160336	
	160782	
13620	136217	A module defined across three files and containing a struct does not compile.
13820	126667	The C++ code generated by the IDL compiler is not scoped correctly if base and derived class methods exist with the same name.

14880	138279	`long long` is silently mapped to `CORBA::Long`.
	140493	`long long` support added for this release.
	167305	
15261	123673	Iinserting a recursive union into an Any causes a 'bad TypeCode' error.
	145321	
	123167	
16180	126210	TypeCodes cause NPR and server core dumps.
16840	131972	The IDL compiler pre-processor does not insert a new line after including a file.
	145164	
	197652	

17140	143191	The following generated code does not compile:
		`void B::IProxyFactoryClass::baseInterfaces (_IDL_SEQUENCE_string& seq) {`
		`add (seq, B_I_IR);`
		`(A::IProxyFactoryClass*) this)-> IProxyFactoryClass::baseInterfaces(seq);}`

17820	144778	The IDL compiler crashes with `-N` switch if include file contains exception and is included inside a module.
18420	123842	Incorrect code is generated for typedefs of arrays.
18520	145890	## preprocessor directive does not work.
19320	147086	Arrays of typedefed types do not compile.
20340	148121	Problem with `#pragma` prefix and user exceptions.
	148464	In the stub code, when `strcmp()` is called to find the ID of a `UserException` raised, it fails and an unknown exception is raised. The `strcmp()` now compares against the exception name that includes the prefix information.
25280	158772	The IDL compiler attempts to process # in an included file even if it is commented out.
28480	143205	Problem with inheritance specification generated for BOAImpl classes.
	170931	
28880	160600	Sequences of two-dimensional arrays fail to build properly when defined within an interface.
33660	185070	The `-N` option fails for exceptions and modules.

	188434	
	214860	
33820	185521	Any insertion `operator <<= obj_ref _var` Any does not work in Orbix 2.3 MT on HP-UX with either the native or ANSI compilers.
35000	188687	IDL does not generate correctly scoped code with `-S` switch when using modules and `enum`.
37461		Generated code does not compile if a fixed size structure is used in IDL.
43200	206096	IDL containing reopened modules does not compile on NT
46880	210779	Including modules from a separate IDL file causes compile errors in generated code.

DSI/DII

Incident ID	PR Number	Synopsis
11780	111029	When trying to set DII request parameters using the `encodeCharArray()` API, the array values are not sent correctly.
	112126	
	136928	
10701	130891	`CORBA::Typecode::equal()` causing memory leak.

IFR

Incident ID	PR Number	Synopsis
268	17667	`readifr` cannot distinguish between structures and exceptions.
347	169199	`putidl` does not have a way of including IDL fields from elsewhere other than the current directory.
	24809	
	168078	
545	204134	When putting IDL into the IFR, the `putidl` utility does not take into account the IDL that already exists in the IFR.
10222	118779	The IFR crashes when being reloaded by `readifr` when `-I` is used with `putidl`.
10680	137066	Problem with null TypeCodes returned.
	144914	
	146203	
	148824	

	152703	
	158500	
	112424	
	122744	
	124676	
	129764	
	141670	
	144221	
	162955	
	165268	
	174499	
	185959	
12105,18180	122554	Cannot use `putidl` with a `const` typedef.
	142576	
	146380	
	193362	
12180	127405	Union causes `putidl` to core dump if the discriminator is an `enum`.
	116042	
	129191	
	143033	
	147998	
	155829	
15342	136930	IFR does not return typedefs correctly.
19821	121851	Two enumerated types with the same elements, but with different type names and scoped differently, do not work.
27541		Interface with union as in parameter cannot be placed in IFR.
28721	170897	Cannot have 'identifier' as an attribute when using `putidl`.
38220	203214	The IDL compiler generates a different TypeCode to the Interface Repository.
	196286	
39465	199305	Cannot use `putidl` with an IDL file that assigns a const value to a typedef type.
47320	211249	`readifr` on UNIX gives a `BusError` after successfully reading the IFR.

Runtime and Protocol Layer

Incident ID	PR Number	Synopsis
125	108680	`_bind()` core dumps if no daemon is present.
479	24102	Incorrect type IDs occurring in generated IORs for `LOCATION_FORWARD` message.
	153350	
	175890	
	175938	
	179851	
625	113847	Use of `_marker()` does not affect the string IOR returned from `object_to_string()`.
	121375	
	139390	
10122	30154	Using `_bind()`, capitals or aliases in the host parameter hang the first server that communicates using IIOP.
	119658	
	146763	
	111224	
	116862	
	135727	
	141076	
	2014898	
10460	148067	Using Orbix 2.3 on Windows NT, a struct in an Any leaks memory.
	163419	
	130726	
	132681	
13600	102561	An infinite loop is caused by re-use of a bogus channel.
	107905	
	190438	
13601	128057	Memory leak with Any type.
	142071	
	137878	
	139464	
13621	114310	Client stubs compiled into a DLL hang when application exits.

	117050	
	117865	
	119392	
	127517	
	136226	
	134903	
	143372	
	146646	
13800	125128	A user exception with an object reference member causes the client to crash.
	154122	
14001	111059	A memory leak occurs when an Any is inserted into an Any.
	140131	
	186421	
	187306	
14200	27583	Anys in exceptions are not marshaled correctly.
14201	111256	There is an interoperability issue with HP OrbPlus because OrbPlus uses : in object keys.
11800	116851	`CORBA::TypeCode::equal()` only does a structural comparison of structs.
12340	130419	Cannot pass an object reference in a structure.
	140701	
	144060	
	152303	
	197538	
15261	123673	Inserting a recursive union into an Any causes a bad TypeCode error.
	145321	
	123167	
19460	147328	A client receives the old marker name in IOR even if the marker name has been changed.
	167489	
19800	148195	If an OrbixWeb client uses `_is_a()` C++ server does not call `inRequestPostMarshal()` filter point.
21360	146245	Anys leak memory when using IIOP and complex types.
22440	149863	Foreign IORs are handled incorrectly by Orbix 2.2 and 2.3.

	181413	
23980	156691	The IOCallback close operation on the client side is not called when using Orbix 2.3c single-threaded.
26860	159384	Unmarshaling a sequence of structs on the client side over IIOP fails.
	186711	
	204422	
	205939	
27620	159149	The client of an unshared server is unable to bind to a second object using markers.
28800	157851	Anys cause memory leaks on HP-UX and Windows NT.
29480	106398	Catching `CORBA::COMM_FAILURE` does not catch `COMM_FAILURE 10081` exception.
	122722	
	172791	
	183155	
29960	162944	On receiving an error some channels may not be shut down correctly.
30060	136540	`narrow()` and `release()` of the same object reference is not thread-safe.
33680	174210	Orbix Wonderwall and Orbix2.3 must be on same machine.
	177854	Orbix 2.3 will not use the hostname in an IOR (object reference). Instead it uses the hostname contained in the object key. This means that if you intend to use Orbix clients to contact Orbix or OrbixWeb servers behind Orbix Wonderwall, with proxified IORs, you must run the Wonderwall and server on the same host, using different ports.
	181394	
	186810	
	187275	
	195541	
	204580	
	214707	
35500	213572	Server hangs when client uses method referring to previously deleted object reference instead of raising an exception.

Orbix Daemon

Incident ID	PR Number	Synopsis
72	15510	The well-known port for a server is not used by the Orbix daemon.
	24207	
487		`rmit -marker` causes the daemon to core dump.

502	21781	Daemon crashes with `rmit -marker` and cores the daemon.
	23566	
	25340	
11741	129002	`orbixd` daemon crashes when there are more than 512 entries in the top level of the Implementation Repository.
13161	135655	`rmit –marker` corrupts the Implementation Repository.
	166139	
13500	145850	Method `IT_daemon::removeUnsharedMarker()` is corrupting the Implementation Repository file.
	145877	
	125046	
	126132	
	145740	
18760	143842	Orbix2.3c daemon can not recognize manually launched server process.
29180	169714	`-per-method` activation mode does not work.
31580		Daemon slow when overloaded. This causes clients to hang and get exceptions.

Orbix Utilities

Incident ID	PR Number	Synopsis
26480	161500	`lsit -R` causes segmentation fault on HP-UX.
30080		Orbix daemon problem with `mkdirit`.

Known Problems, Workarounds and Tips

The known problems, workarounds and tips for Orbix 3.0 are as follows:

Installing on HP

In order to change the location of Orbix, you must use **Change Product Location** in the **Action** menu and not **Change Target**, to insure that the configuration steps are implemented correctly.

DII

The DII does not handle user exceptions.

Runtime and Protocol Layer

Unshared activation mode does not work with IIOP.

Throwing exceptions with object references causes a core dump.

Utilities

`putit -marker` is not implemented for IIOP.

Unsupported Types

Orbix does not support `long double`, `wstring` or `wchar`.

Fixed Type

HP-UX cfront compile error occurs when template-based destructors are being called. Reproduction can be seen in cases of IDL arrays and sequences of fixed types.

The generated code does not compile on HP-UX cfront due to the explicit call to the fixed type's destructor, which is based on a template class. HP-UX cfront cannot compile calls to template-based class destructors. This is a bug in the compiler itself.

Overview

Consider the following template class:

```
template<short, short> class X {
    X () {};
   ~X () {};
 }
```

An instance of this class could be:

```
X<10,5> myX;
```

There are three valid ways of calling the destructor, outlined as follows:

- `myX.X<10,5>::~X<10,5> ();`
- `myX.~X<10,5> ();`
- `myX.X<10,5>::~X ();`

The standard generated code currently uses first choice. This works fine but cfront can not compile it. The only compilable option for cfront is `myX.~X();` which is actually illegal code.

Signal Handling

Orbix does not include a signal handler for SIGPIPE.

Any

Unions with discriminators of type `long long` and `unsigned long long` do not work within CORBA::Any.

On HP-UX, sequences of interfaces do not work correctly with CORBA::Any. To work around this problem, use sequences of `CORBA::Object` instead.

Orbix Code Generation Toolkit

The Orbix Code Generation Toolkit is a powerful, new component of Orbix. It speeds up application development by automating many coding tasks.

The Orbix Code Generation Toolkit consists of the following:

- An executable, called *IDLgen*, which is a code generation engine.
- Bundled code generation scripts, called *genies*. These genies instruct IDLgen what kind of code to generate. For example:

  The C++ genie (`cpp_genie.tcl`) can generate a C++ client/server application for a specified IDL file.

  The HTML genie (`idl2html.tcl`) translates an IDL file into its HTML equivalent.
- Standard libraries for use with IDLgen. For example, there is a library that maps IDL constructs into their C++ equivalents. You can use these standard libraries to help you write your own genies.

Compatibility with Other IONA Products

The Code Generation Toolkit is new to Orbix 3. However, an effort has been made to make it work with previous versions of Orbix, notably versions 2.2 and 2.3. If you wish to use the Code Generation Toolkit with Orbix 2.2 or 2.3, edit the configuration file for IDLgen (its default location is `<IONA-ROOT>/config/idlgen.cfg`) and change the following entries:

1. Set the `default.orbix.version_number` entry to the value `2.2`, or `2.3`.
2. Set the `default.orbix.install_root` entry to the directory in which Orbix 2.2 or 2.3 is installed.

Recursive `structs` and recursive unions are `structs/unions` that have an embedded sequence of themselves; the following is an example of a recursive `struct`:

```
struct node {
       long          data;
       sequence<node> children;
};
```

Some patches for Orbix 2.3 on some platforms introduced a change in the way that recursive `structs/unions` are implemented. The C++ genie must generate slightly different code for recursive types, depending on whether or not you are using a version of Orbix with an affected patch.

If the C++ genie generates code for recursive types that does not compile with a C++ compiler then it is likely that you have a version of Orbix 2.3 with the affected patch. In this case, add the following line o the IDLgen configuration file:

```
default.cpp.nest_recursive_type_seq_inside_type = "1";
```

You can do this easily by going to the end of the IDLgen configuration file, and uncommenting the line that contains this setting.

This change to the IDLgen configuration file needs to be performed only if:

1. You are using a patched version of Orbix 2.3.
2. You use recursive types in your IDL files.
3. The C++ code produced by the C++ genie for these recursive types does not compile.

You do not need to make this change if you are using Orbix 2.2 or 3.0, or if you are using 2.3 and are not experiencing any problems compiling code generated by the C++ genie.

Documentation

The Orbix user documentation has been updated for this release. A new edition of the following manual is included with the product:

- *Orbix Code Generation Toolkit Programmer's Guide*

A Simple Code Generation Toolkit Example

To see an example of the power of the Code Generation Toolkit:

1. Create a new directory and copy any IDL file into it. For example, consider an IDL file called `foo.idl`.
2. Open a command window for the directory which contains the IDL file and type in the following commands:

```
idlgen cpp_genie.tcl foo.idl -all
nmake
nmake putit
client localhost
```

The first line runs the C++ genie on your IDL file. It generates the source code for a complete client/server application together with a Makefile.

The second line uses the generated Makefile to compile your client/server application.

The third line uses the `putit` target in the generated Makefile to register the server with Orbix (make sure that the Orbix daemon is running before executing this command).

The final line runs the client application. It takes one command-line parameter that is the name of the host where the server is running.

You can find more information about the C++ genie in the *Orbix Code Generation Toolkit Programmer's Guide*.

Known Problems, Workarounds and Tips

This section summarizes known problems, workarounds and tips with the Orbix Code Generation Toolkit. A list of the known limitations of IDLgen is provided in Chapter 1 of the *Orbix Code Generation Toolkit Programmer's Guide*.

Microsoft Windows NT

The following issues are relevant to the use of the Code Generation Toolkit on Windows NT.

Compiler Version

This product has been designed to work with Microsoft Visual C++ v6.0, Service Pack 2. This is the only compiler environment supported for this product.

Orbix C++ Client/Server Wizard v1.0

The Orbix C++ Client/Server Wizard is a graphical tool for Microsoft Visual C++. It is intended for use with Visual C++ version 6.0.

The Orbix Code Generation Toolkit install procedure copies the wizard into your Developer Studio installation. To run this Wizard:

1. Launch Developer Studio.
2. Select **File→New**.
3. Select the **Projects** tab.
4. Select the **IONA Orbix C++ Client/Server Wizard** option.

The wizard generates full client or server code for you, based on your OMG IDL definitions. At present you cannot create both a client and a server simultaneously. If you want both client and server code, simply run the wizard twice.

You can access context-sensitive help at any time while using the wizard by pressing the **F1** key or by selecting the **Help** buttons.

The wizard makes use of two Tcl files, supplied with your installation of the Orbix Code Generation Toolkit. They are called `cppwpre.tcl` and `cppwfull.tcl` and are placed in this directory:

```
<IONA-ROOT>\IDLgen3.0\genies\wizards
```

Do *not* modify these files in any way, as they are used internally by the wizard. IONA cannot support the wizard if these files are changed in any way.

Also note that your include and library directories in Visual Studio must be set up to point to your Orbix 3.0 installation. To do this:

1. In Developer Studio, select **Tools→Options**.
2. Select the **Directories** tab.
3. Add your Orbix 3.0 `include` and `lib` directories to the appropriate lists.

Manually Installing the Orbix C++ Client/Server Wizard v1.0

If you did not install Visual C++ 6 before installing Orbix 3 then the installation of the Orbix C++ Client/Server Wizard is incomplete. Two wizard files – `it_cppwiz.awx` and `it_cppwiz.hlp` will have been placed in the `<IONA>\IDLgen3.0` directory.

Once you have installed Visual C++ 6, you can complete the installation of the wizard by copying these two files into the correct directory within your Visual C++ installation. The correct directory is:

```
<DevStudio>\Common\MsDev98\Template
```

The wizard should now appear in the **New Projects** listing of the Visual C++ IDE.

Licensing

You must install the license key supplied to you in order to use the Orbix Code Generation Toolkit. You can install the key with the executable `licence.exe`, which is in the `bin` subdirectory of the Orbix installation directory (default `C:\IONA\bin`).

The executable file to be licensed is named `idlgen.exe`, and can be found in the same `<IONA-ROOT>/bin` directory.

Using the Microsoft VC++ Command Line Compiler

The Orbix Code Generation Toolkit makes use of the Microsoft Visual C++ Command Line Compiler. In order for this compiler to run correctly, certain environment variables must be set.

A batch file, named `VCVARS32.BAT` is provided with the VC++ compiler, in the directory `DevStudio\Vc\bin`. You must execute this batch file to set the necessary environment variables for operation of the command line compiler. Please refer to the Microsoft VC++ documentation for further information.

OrbixCOMet Desktop

This document describes the changes made to the OrbixCOMet Desktop product in the 3.0 release.

Development Environments

OrbixCOMet Desktop is not supported on versions of Windows NT earlier than 4.0. This is because DCOM is not available on those versions.

OrbixCOMet Desktop supports both the Automation/CORBA mapping and the COM/CORBA mapping as specified in the COM/CORBA Interworking Document (*ORBOS-97-09-01*).

OrbixCOMet Desktop 3.0 has been tested with Automation client applications built with the following:

- PowerSoft PowerBuilder Version 6.0
- Borland Delphi 3/4

  When using Delphi4, Inprise recommend that you make a call to `Application.Initialise();` before making any COM calls. This includes any calls to OrbixCOMet.
- Microsoft Visual Basic Version 5.0 (SP3)
- Microsoft Visual Basic Version 6.0
- Microsoft Visual C++ 6.0 (SP2)
- Microsoft Excel97
- Microsoft Internet Explorer 4.0 or higher with VBScript

OrbixCOMet Desktop 3.0 has been tested with COM client applications built with the following:

- Microsoft Visual C++ 6.0 (SP2), MIDL Compiler Version 5.01.0164

OrbixCOMet Desktop 3.0 has been tested with CORBA server applications built with the following:

- Orbix3.0
- OrbixWeb3.1c, (using JDK 1.1.x)

OrbixCOMet Desktop 3.0 has been tested with CORBA client applications built with the following:

- Orbix3.0
- OrbixWeb3.1c, (using JDK 1.1.x)

Installation Issues

You should uninstall any previous version of OrbixCOMet before installing this version. If you are installing OrbixCOMet as part of the Orbix 3.0 package, you must first install Orbix 3.0 and reboot your machine before running the

OrbixCOMet `setup` program. If you have purchased OrbixCOMet as a stand-alone component, the Orbix runtime DLLs required by OrbixCOMet will be installed by the OrbixCOMet `setup` program if necessary.

OrbixCOMet shipped within the Orbix 3.0 package does not install the Orbix runtime.

If you have a licence code for a full release of OrbixCOMet, you must supply it during the installation process. If you are currently using an evaluation version, you can leave the licence code field blank and this installation will default to an evaluation that is valid for 21 days from the date of installation.

If the installation program crashes during the DLL registration, or it reports that various DLLs cannot be found, you can manually register the OrbixCOMet runtime using the batch file `regCOMet.bat` located in *<COMetROOT>*`\bin`.

Knowledge Base and Support

OrbixCOMet support is provided in the form of a Knowledge Base located at:

```
http://www.iona.com/online/support/kb/OrbixCOMet/index.html
```

You can also purchase a separate support contract that entitles you to email-based support queries. (Contact sales@iona.com for more details.) In addition, you can subscribe to a peer mailing list, comet-users@iona.com, by sending an e-mail to comet-users-request@iona.com with the word "subscribe" in the body of the message. If you want to unsubscribe, do the same but use the word "unsubscribe". If you encounter any problems, you should report them to users@iona.com.

New Features

This section describes the new functionality and major changes added in the OrbixCOMet Desktop 3.0 release.

Common Demonstrations

Common demonstrations are included to illustrate out-of-the-box interoperability with Orbix 3.0. You can find the common OrbixCOMet client demonstrations in:

```
\iona\demos\AnyDemo\COMet\VB6
\iona\demos\BankExceptions\COMet\VB6
\iona\demos\Bankinherit\COMet\VB6
\iona\demos\banksimple\COMet\VB6
\iona\demos\banksmartproxy\COMet\VB6
\iona\demos\callback\COMet\VB6
\iona\demos\Grid\COMet\VB6
```

Each `.exe` file in the `\VB6` directory is an OrbixCOMet client of the `corba` server found in the *<demoname>*`\cxx` directory.

Before you run any of the OrbixCOMet common demonstrations, ensure that the IFR is registered with the Orbix daemon. You should also ensure that the necessary OMG IDL files are registered in the Interface Repository. You can do this by running the `PutAllIDL.bat` file in the `iona\demos` directory.

New Configuration Format

In keeping with Orbix 3.0, OrbixCOMet 3.0 uses a new configuration format. OrbixCOMet no longer stores any configuration information in the Windows registry. Instead, all configuration values are contained in `orbixcomet30.cfg` located in `\iona\config`. All configuration values now use scoped names (for example, `COMet.Mapping.UseSAFEARRAYMapping`). All new code should use this scoped format when setting/getting configuration values. For the purposes of backwards compatibility, OrbixCOMet 3.0 will accept the following set of unscoped names:

```
// COMet.Config
"COMET_HANDLER_LOCATION"     ,
"COMET_DEFAULT_PROTOCOL"     ,
"COMET_DAEMON_HOST"          ,
"COMET_ROOT"                 ,
"COMET_SHUTDOWN_POLICY"      ,
"COMET_UPDATE_LEVEL"         ,
"COMET_SMART_STACK"          ,
"COMET_PRE30_FORMAT"         ,

// COMet.Mapping
"UseSAFEARRAYMapping"        ,
"SAFEARRAYS_CONTAIN_VARIANTS",


// COMet.Debug
"MessageLevel"               ,

// COMet.TypeStore
"TYPEMAN_CACHE_FILE"         ,
"TYPEMAN_DISK_CACHE_SIZE"    ,
"TYPEMAN_MEM_CACHE_SIZE"     ,
"TYPEMAN_IFR_HOST"           ,
"TYPEMAN_IFR_IOR_FILENAME"   ,
"TYPEMAN_IFR_NS_NAME"        ,
"LOG_CACHE_HITS"             ,
"LOG_DELETES"                ,
"TYPEMAN_LOGGING"            ,
"TYPEMAN_LOG_FILE"           ,
"LOG_TYPEMAN"                ,
"TYPEMAN_READONLY"           ,

// COMet.Licensing
"IT_KEY"                     ,
// Common
"IT_INT_REP_PATH"            ,
"IT_DAEMON_PORT"             ,
"IT_DAEMON_PROTOCOL"         ,
"IT_DAEMON_SERVER_BASE"      ,
"IT_LOCAL_DOMAIN"
```

You should ensure that all new code adopts the new scoped configuration format. You can browse/set configuration values by using the common configuration tool `\iona\bin\ConfigurationExplorer.bat`.

Support for Coclasses

The OrbixCOMet type store now includes support for coclasses in type libraries.

Scoped Names to Avoid Clashes

When the type store is primed from a type library, all types are scoped with the type library name (for example, `Excel::Application`). This avoids clashes where multiple type libraries contain interfaces or types that share the same name. For example, both the MS Excel and MS Word libraries contain a type called `Application`. When the type store is primed with these libraries, the two are distinguished by using the scoped names `Excel::Application` and `Word::Application`.

Prefixing of Type Names

When priming the type store from a type library, any types whose names begin with a leading underscore will be prefixed with `IT_`. This feature is used by `ts2idl` when generating OMG IDL based on type library information, because leading underscores are illegal in OMG IDL. Interfaces whose names start with leading underscores are commonplace in type libraries, and denote interfaces that are "hidden" (that is, interfaces that should not be displayed by a type library browser such as `OleView.exe`). Repository IDs of these types also contain `IT_` to mask the leading underscore.

Extended OMG IDL Generation

OMG IDL generation by `ts2idl` based on type library information has been extended to ensure that:

1. All mapped OMG IDL interfaces inherit from `CosLifeCycle::LifeCycleObject` to allow CORBA views of COM objects to be destroyed from a client application. Hence all such OMG IDL files will include `lifecycle.idl`. OrbixCOMet implements part of the life-cycle service as mandated by the COM/CORBA specification. However, the only `CosLifeCycle::LifeCycleObject` method that can be called is `remove()`. If you call `copy()` or `move()`, it will result in a `NO_IMPLEMENT` exception, which is a valid response.
2. All mapped OMG IDL interfaces also inherit from `CORBA_COM::Composable` to allow navigation between the different interfaces exposed by DCOM objects.
3. Pseudo coclass interfaces are generated for all coclasses. These provide an alternative, OrbixCOMet-specific way to navigate between the different interfaces exposed by DCOM objects.

For examples of the preceding points 2 and 3 you can refer to the `fortune` demonstration in *<COMET_ROOT>*`\demos\corbaclient\fortune`.

-b Switch to ts2idl

The `-b` switch can be used when generating OMG IDL based on type library information stored in the type store. Its purpose is to attempt to keep the number of generated lines of OMG IDL to a minimum by reducing the set of OMG IDL types that need to be produced.

It specifies that interface pointers that are passed as parameters to operations described in the type library are mapped as type `CORBA::Object` in the generated

OMG IDL, rather than as their "dynamic type". Use this switch in conjunction with the `-r` switch. This can dramatically reduce the amount of IDL that is generated. For example, if you generate full OMG IDL for Excel 97, you will get tens of thousands of lines of output that can take a long time for the IDL compiler and C++ compiler to process. In the case of Excel, there are normally only about five interfaces you really use, such as WorkBook, Worksheet, and so on. This means use of the `-b` option reduces the output to about 3000 lines of IDL.

For an example of its use, refer to the Excel CORBA client in the `demos\corbaclient\excel` directory.

Keyword Prefixing in Generated IDL

To prevent possible compilation errors in IDL compiler-generated code, `ts2idl` now checks the `COMet.Mapping.KEYWORDS` configuration value for a list of keywords that should be prefixed with `IT_clash`. For example:

```
COMet {
  Mapping {
         KEYWORDS = "DialogBox, remove, move, copy";
  };
};
```

In the preceding code, `DialogBox` is included because the MS Excel type library contains a method called `DialogBox`. This causes errors when compiling the C++ code generated by the Orbix IDL compiler, because `DialogBox` is also a macro in a Windows include file. Similarly, `remove`, `move` and `copy` are treated as keywords in case the MIDL interface to be mapped contains methods of these names. Recall that all mapped OMG IDL interfaces will inherit from `CosLifeCycleObject`, and redefinition of OMG IDL operations in derived classes is illegal.

Extended Support for Directly Binding to DCOM Servers

Support for directly binding to DCOM servers has been extended over COMet 1.0 UR2. A new set of demonstrations in the `\iona\comet3.0\demo\corbaclient` directory illustrate the usage of this feature. Included are Orbix C++ clients of MicroSoft Excel 97, Microsoft Word 97, an OrbixWeb client of the simple `fortune` DCOM server, and Orbix/OrbixWeb clients of the `mfccalc` demonstration. The `mfccalc` demonstration is an Automation server written using MFC that implements a simple calculator interface. The server code is not shipped (it is a Microsoft demonstration available with the SDK) to prove that it is possible to interoperate with existing DCOM servers without the need to write one line of code (using `DIOrbixServerAPI`) or any broker generation (via a static bridge).

Moniker Support

When talking to a DCOM server that supports file monikers, a UNC filename can be supplied as a marker to denote a particular instance of a DCOM object that should be activated. For example, to retrieve a CORBA view of an Excel Workbook called `salaries.xls`, shared out `\\advice\root\misc`, you would issue a call to `_bind()` as follows:

```
char * hostName = argv[1];
```

```
Excel::Workbook_var wbVar =
Excel::Workbook::_bind("\\\\advice\\root\\misc\\salaries.xls:
       ExcelSrv", hostName);
```

In this case, the `hostName` denotes the name of the machine where `custsur.exe` has been registered with the Orbix daemon. The filename must be in UNC format, and cannot contain colons (that is, `:`). This is because colons are special characters as far as the preceding `_bind()` call is concerned (used as a separator in the marker/server pair).

-t Switch with custsur.exe

`custsur.exe` now takes a `[-t <timeout>]` switch that can be used when registering it as a CORBA server. It specifies the timeout in milliseconds after which the server can terminate if both of the following conditions apply:

- It has not received an invocation from a client in the previous *<timeout>* milliseconds.
- There are no outstanding references to CORBA views held by clients. In other words, if the CORBA client has not called `CosLifeCycle::LifeCycleObject::remove()` on all object references after it is finished with them, the server will not shut down. If one client calls `remove()`, a separate client that also holds a reference to that object will receive a system exception. This is because the CORBA specification of `remove()` mandates the destruction of the target object on the server side.

The default timeout (that is, if the `-t` switch is not used) is `CORBA::Orbix.INFINITE_TIMEOUT`. This means the server will never be timed out and it should be killed using `killit.exe`.

Use the `-t` switch as follows:

```
putit excelSrv "d:\iona\comet3.0\bin\custsur.exe -t 10000"
```

Note the use of quotes around the server launch command. The preceding example specifies a timeout of 10 seconds (10,000 milliseconds) for the server `excelSrv`.

Generation of IORs via custsur.exe

When exposing DCOM servers to CORBA clients written using other ORBs, `custsur.exe` can be used to generate an IOR for (for example) registration in a Naming Service. For example:

```
custsur -g -s Calculator -i mfccalc::CCCalcDlg -f c:\temp\ior.log
```

This generates an IOR for an object whose interface is `mfccalc::CCCalcDlg` and which resides in the server `Calculator`. In this case, the IOR is dumped in `c:\temp\ior.log`.

Support for CORBA Interfaces Using #pragma Prefix

There is now support for CORBA interfaces that use the `#pragma` prefix. Repository IDs are now supported internally throughout OrbixCOMet.

Enhanced Debug/Trace Information

All logging is now subdivided into various categories (for example, error logs, marshalling logs, and so on).

Support for Callbacks with Complex Types in Automation and COM

There is now support for callbacks with complex types in Automation and COM.

Servers Using DIOrbixServerAPI Dispatch Own Events

Servers that use `DIOrbixServerAPI` must now dispatch their own events.

COMetIFR Integrated with Orbix IFR

There is now only one IFR because the COMetIFR has been integrated with the Orbix IFR. Refer to "Stand-Alone IFR" in this document for more details.

Single Command for Reading IFR into Type Store

This simplifies priming of the type store. The following command:

```
typeman –e *
```

will read the entire IFR into the OrbixCOMet type store cache. If the IFR is not too large, this will not take too long, and it obviates the need to keep everything in modules (for example, miscellaneous global typedefs and constants).

Read-Only Mode in Type Store

If OrbixCOMet is to be deployed in an environment where there might be multiple clients to the type store (for example, with DCOM-on-the-wire), for safe operation the type store must be fully primed and then made read-only via the following configuration file setting:

```
COMet.TypeMan.TYPEMAN_READONLY = "yes"
```

Using the Naming Service to Locate the IFR

Support has been added for location of the IFR via the Naming Service. This means load balancing of IFRs is now possible, along with interoperability with the Interface Repositories of other vendors.

To use this feature:

1. Ensure that the Naming Service is correctly installed. For example:
   ```
   putit NS c:\iona\bin\NS.exe
   ```
2. Add configuration file entries as follows:
   ```
   COMet.TypeMan.TYPEMAN_IFR_NS_NAME = "config.ifr"
   COMet.TypeMan.TYPEMAN_IFR_IOR_FILENAME = "c:\temp\typeman.ior"
   COMet.Services.NameService = "c:\temp\names.ior"
   ```
3. Start the Orbix daemon required by the Naming Service. For example:
   ```
   c:\>start orbixd
   ```
4. Start the IFR using `startIFR.bat`. For example:
   ```
   c:\>startIFR 1234
   ```
   This writes out a valid IOR for the currently running instance of the IFR.
5. Start the Naming Service and get it to write out its IOR. For example:
   ```
   c:\>start NS –I c:\temp\names.ior
   ```
6. Run `FirstPutName.bat`. For example:
   ```
   c:\iona\comet_3.0c\bin>FirstPutName
   ```
   This gets the IOR to the IFR and puts it in the Naming Service, making it available to OrbixCOMet's `TypeMan` component. If OrbixCOMet is not installed in `c:\iona\comet_3.0c\bin`, you should run `FirstPutName.bat` from the `\bin` directory wherever OrbixCOMet is installed.

To use the Naming Service subsequently, use `PutName.bat` instead of `FirstPutName.bat`. For example:

```
c:\iona\comet_3.0c\bin>PutName
```

If OrbixCOMet is not installed in `c:\iona\comet_3.0c\bin`, you should run `PutName.bat` from the `\bin` directory wherever OrbixCOMet is installed.

IORs to CORBA servers are only valid if generated by the currently running instance of the server. This means if the IFR is closed down, its IOR must be regenerated the next time it is started up using `startIFR.bat` and then added to the Naming Service using `PutName.bat`. `PutName.bat` assumes that the Naming Service is already running.

Bugs Cleared in this Release

This section describes the incidents cleared in this release. All incidents are cross platform unless otherwise stated. The incidents are described in terms of **Incident ID**, **PR Number**, and **Synopsis**, as described on page 20.

Incident ID	PR Number	Synopsis
14080		OrbixCOMet uses `NS` as tag, but this is normally used as a server name. This tag has been changed to `NAME_SERVICE`.
27266		Some of the CORBA servers throw an exception on exiting demonstrations.
27280		Demos - VB - bank client - Windows 95 - COMet Bridge location fails.
29582		Mapping for Automation unions not implemented.

Incident ID	PR Number	Synopsis
33520		Problem using return, `out`, `inout` (in some cases) parameters and all complex types with CORBA client against COM servers.
35060		Passing a sequence of structs in an any does not work. Also applies to an any contained within a struct.
38800		A complete example should be used when showing exception handling.
39080		Exception handling does not work with COMet and Delphi3.
41700		Bounded sequences in COM mapping.
47680		Callback using complex types are broken in Update Release 2 (UR2).
47702		Marshalling error message with the bank server demo.
51366		Incomplete COMet deregistration.
51436		Problems with sequences of sequence of strings
47660		Marshalling error message with the bankserver demo shipped with OrbixCOMet UR2.
48000		VB client cannot bind to a DCOM server using `GetObject()`. Fixed for **IDispatch-based** clients.
48640		There are problems with `SafeArray` mapping and OrbixCOMet UR2.
50840		Marshalling error occurs when `it_default` is called in a C++ CORBA client of a DCOM server.
51160		The wrong exception is created when the `objectName` parameter of `GetObject` is invalid.
51409		`ts2tlb` generates error in OrbixCOMet UR2 but not in UR1.
51437		`ts2idl` generates rubbish characters.
51451		`README.txt` is incorrect for the VB standalone demo.
27221		Demos - VB5 - `banksrv` - No explanation what it does or how it works.
27261		Demos - COM - `cocreate` compiles but `aliassrv` fails.
27263		README.txt errors in demos.
27267		Problem with sequences in demos on Windows 95 and NT.
29560		Cannot pass a sequence of object references.
29562		Reference counting error in collection objects.
29580		Reference counting error.
29584		Crash on exit caused by DSI timer.

Incident ID	PR Number	Synopsis
33640		Callbacks do not work with OrbixCOMet.
40960		`ts2tlb` causes error when IDL operations pass type `Object`.
40961		`ts2idl` does not handle coclasses.
40981		You cannot easily `_bind()` to a COM server without using `ITServerAPI`.
41000		`typeman` command line utility crashes occasionally with `-c` (contents) option.
41021		Typestore doesn't handle COM CoClasses.
41022		No scoping is available to differentiate between objects in different (imported) type libraries.
41042		Imported type libraries must be added to type store individually, and in correct order—this is now done automatically.
41043		Multi-parameter attributes in COM are not represented correctly.
41044		Hidden attributes and methods in COM are not represented correctly.
41060		Inheritance in type libraries is not correctly reflected in generated OMG IDL.
41061		Type libraries cannot be located by their guids instead of their paths.
41062		Some objects in typelibs cannot be located by `typeman` lookup.
41300		You cannot use scoped interfaces names with the IOR tagged format string to `GetObject`.
41301		You cannot lookup services using the '.' format version of the `GetObject` string.
41500		The const values for `EXCEPTION_USER` and `EXCEPTION_SYSTEM` are reversed.
44260		`typeman` crashes when reading a union from the COMetIFR.
51640		OC* files created in temp directory when using duals/or watch window in VB.

Documentation

The OrbixCOMet 3.0 documentation consists of the following:

- *OrbixCOMet Desktop Getting Started*
- *OrbixCOMet Desktop Programmer's Guide and Reference*
- `COMetIntro.exe` (a multimedia OrbixCOMet presentation)

Known Problems, Workarounds and Tips

This section summarizes known issues and tips relating to the OrbixCOMet 3.0 release.

Licensing Issues

OrbixCOMet Desktop 3.0 requires a valid license to function correctly. During installation you are given the opportunity to enter a license key. If you choose not to do so, you are granted an evaluation license that is valid for 21 days from the date of installation.

During that time you will be notified periodically that you are using an evaluation copy. If you receive such a reminder, simply re-run your application to continue.

The following messages mean the license key has been corrupted/deleted. The workaround is to reinstall OrbixCOMet:

```
"COMet Licensing error: Invalid License Key format"
"COMet Licensing error: Missing License Key"
```

The following message means the product has deactivated itself until a valid, up-to-date licence has been obtained from IONA (via sales@iona.com):

```
"COMet Licensing error: License Key has expired"
```

The following message means the OrbixCOMet licensing server could not be found:

```
"COMet Licensing error: Missing License DLL \ (Have you registered
ITLicense.DLL?)"
```

The workaround to the preceding message is to ensure that `ITLicense.dll` and `it_licps.dll` are both registered as follows using `regsvr32.exe`:

```
regsvr32 ITLicense.dll
regsvr32 it_licps.dll
```

If any of these errors occur, you should try to either:

- Re-license the product with a valid licence obtained from IONA.
- Re-register the two license DLLs already described.

The following message appears approximately every 50 runs of an OrbixCOMet application and it provides information about how to purchase a full licensed version of OrbixCOMet if you so desire:

```
"COMet Eval-License Reminder : .... "
```

If you subsequently receive a full OrbixCOMet license from IONA, you should enter the license code in the `"COMet.Licensing.IT_KEY"` entry within the `orbixcomet30.cfg` file.

Backwards Compatibility with the Orbix/ActiveX Integration

This section documents some differences between OrbixCOMet and the Orbix/ActiveX Integration.

Compliance Issues

OrbixCOMet Desktop is designed to be backwards compatible with IONA's previous Automation/CORBA bridge. This is subject to changes to the standard

interfaces/mappings as laid down in the COM/CORBA Interworking RTF. This gives rise to the following issues:

- `DICORBAObject`

  Return values are now `VARIANT_BOOL`s rather than `BOOLEAN`s for appropriate methods.

- DIForeignObject

  `GetRepositoryId` method has been renamed to `GetUniqueId`.

- `CORBA::Boolean` in OMG IDL maps to `VARIANT_BOOL` in Automation.

- `ITstdInterfaces.tlb` replaces `itole.tlb` as the type library containing Automation/CORBA standard types.

- Addition of new standard interfaces.

- Some methods and properties on standard interfaces have been deprecated.

Sequences

OrbixCOMet supports two Automation mappings for sequences and arrays. These are to Automation SAFEARRAYs and Automation Collections. Two mappings are required because not all Automation controllers support Automation SAFEARRAYs. (for example, PROGRESS Software tools, PowerBuilder, and so on).

You can select which mapping is active in your application by making a call to `orb.SetConfigValue("COMet.Mapping.UseSAFEARRAYMapping", <value>)` where *<value>* is either set to "yes" or "no". Refer to the *OrbixCOMet Desktop Programmer's Guide and Reference* for details about this and other OrbixCOMet configuration settings.

You can alter the default mapping in effect for your machine by modifying the configuration entries found in `\iona\config\orbixcomet30.cfg`.

In the case of sequences, the old Orbix static bridge mapping assumed that the sequence names would be in the following format:

```
_IDL_SEQUENCE_long
_IDL_SEQUENCE_15_MOD1_IFACE1_STRUC
```

for the following OMG IDL:

```
module MOD1
{
  interface IFACE1 {
         struct STRUC {
                long l;
         };
                typedef sequence<long> longSeq;
         typedef sequence<STRUC,15> STRUC15Seq;
         void op1(in longSeq LS, in STRUC15Seq SS);
  };
};
```

Orbix Desktop was unable to create types based on their typedef names and it required the `_IDL_SEQUENCE_xxx` version to be passed to `CreateType()`. These `_IDL_SEQUENCE_xxx` names are artifacts of the IDL C++ mapping. They are therefore not considered real type names by either OrbixCOMet or the Interface Repository.

OrbixCOMet does not have this typedef restriction.

If you have problems using older static bridge code with references to `_IDL_SEQUENCE_xxx` names, you can manually prime the type store with information for the required sequence typedef name (that is, the real name of the `_IDL_SEQUENCE_xxx` name that is passed to `CreateType()`).

The `typeman.exe` utility supplied with the bridge will do this for you. To prime the cache with information about a type, use the following command:

```
typeman -e <type name>
```

This looks up the cache to see if there is an available entry for *<type name>*. If there is none, it contacts the Interface Repository (default local machine) and retrieve the type information.

Note: If the Interface Repository is located on a remote machine, its remote machine name can be specified via the configuration entry `COMet.TypeMan.TYPEMAN_IFR_HOST` located in the registry under `HKEY_LOCAL_MACHINE\Software\IONA Technologies\DCOM bridge\1.0c\Config`.

The process of priming the cache with the sequence typedef name will have the affect of generating the necessary backward compatible alias names. For example, an entry for `MOD1::IFACE1::STRUC15Seq` will create the correct alias `_IDL_SEQUENCE_15_MOD1_IFACE1_STRUC` in the type store.

If you supply a top-level module name such as the following:

```
typeman -e MOD1
```

it will be sufficient to resolve all backwards compatibility sequence issues for all types in the `MOD1` module. This is only an issue for sequences in use by existing static bridge applications. The cache is normally self-managing.

Any new code being developed should use the correct typedef names when making calls to `CreateType()`. Existing code can be migrated to the new format names over time. Support for the older `_IDL_SEQUENCE_xxx` names might be deprecated with a future release of OrbixCOMet.

Another alternative would be to manually change calls to `CreateType()` to use the correct name as specified in the IDL file. For example, rather than the following:

```
objFactory.CreateType(Nothing,
                      "_IDL_SEQUENCE_15_MOD1_IFACE1_STRUC")
```

use the following:

```
objFactory.CreateType(Nothing,"MOD1/IFACE1/STRUC15Seq")
```

The scope separator is indicated by a forward slash (that is, '/').

Winidl/Brokers

OrbixCOMet does not support the `Winidl` wizard previously available with Orbix Desktop. Neither does it generate brokers of any kind. Such issues are related to the implementation of IONA's COM/CORBA bridging technology. They will not affect Visual Basic, PowerBuilder and other client code. Any issues that do arise are bugs in the OrbixCOMet compatibility support and should be reported as such.

Protocol

OrbixCOMet supports both the Orbix protocol (POOP) and OMG IIOP. The choice of protocol is determined by the configuration entry:

```
COMet.Config.COMET_DEFAULT_PROTOCOL="IIOP"
```

in the `orbixcomet30.cfg` file.

Valid values for the entry are `"POOP"` and `"IIOP"`. One example of why you might wish to use POOP is if you are using a pre-Orbix2.3 version of the Orbix daemon.

If you experience intermittent problems connecting to the Orbix 2.3c daemon, set the default daemon protocol to IIOP. You can do this by adding a configuration setting as follows in the `orbixcomet30.cfg` file:

```
COMet.Config.IT_DAEMON_PROTOCOL="IIOP"
```

Supported Mappings

The following mappings are supported by this release of OrbixCOMet:

- Bi-directional Automation/CORBA as per the *COM/CORBA Interworking Specification*, OMG Document *ORBOS/98-02-01*, (February 01 1998).
- Bi-directional COM/CORBA as per the *COM/CORBA Interworking Specification*, OMG Document *ORBOS/98-02-01*, (February 01 1998).

Usage Models

The following usage models are supported for Automation by this release of OrbixCOMet:

- In-process dispatch
- Out-of-process dispatch
  - Local machine (IIOP-on-the-wire)
  - Remote machine (DCOM-on-the-wire)
- In-process dual interface
- Out-of-process dual interface (local/remote machine)
  - Local machine (IIOP-on-the-wire)
  - Remote machine (DCOM-on-the-wire)

The following usage models are supported for COM by this release of OrbixCOMet:

- In-process COM custom interfaces
- Out-of-process COM custom interfaces (local/remote machine)
  - Local machine (IIOP-on-the-wire)
  - Remote machine (DCOM-on-the-wire)

OrbixCOMet Desktop is a bidirectional dynamic bridge. This means it supports:

- COM/Automation clients of CORBA servers.

- Callbacks (that is, invocation from a CORBA server upon a COM/Automation client).
- Implementing CORBA servers in Visual Basic, PowerBuilder, and so on, using the `IT_ServerAPI` interface. For an example of how to do this, refer to the sample application in the *<COMET ROOT>*`\demo\vb6\bankSrv` directory.
- CORBA clients of native DCOM servers (for example, MS Excel, MS Word, and so on). For examples of this, refer to the sample applications in the *<COMET ROOT>*`\demo\corbaclient` directory.

Known Issues

The following are known issues:

- If using OrbixCOMet on Windows 95, the Orbix 3.0 runtime used by OrbixCOMet requires Winsock 2 DLLs that are not, by default, installed on Windows 95 machines. (Winsock 2 is supported on Windows 98 and NT 4.) Windows 95 users must download an update. (Winsock 2 was not in Win95, Win95 SP1, or Win95 OEM SR2). The download plus installation instructions are available from:

  `http://www.microsoft.com/windows95/downloads/contents/wuadmintools/s_wunetworkingtools/w95sockets2/default.asp`
- Marshalling interface pointers across apartment boundaries when using the bridge in-process. Using the bridge out-of-process is fine.

  This is only relevant if the bridge objects are instantiated in a COM single-threaded apartment. Using OrbixCOMet objects in a free-threaded apartment is fine.

  When using OrbixCOMet in C++, you should create a multithreaded apartment. For example:

  ```
  CoInitializeEx(0, COINIT_MULTITHREADED);
  ```
- There is a problem with Visual Basic keeping DLLs loaded in memory even after the application has terminated. This causes OrbixCOMet to prematurely execute its shutdown procedures in response to a positive result to `CoFreeUnusedLibraries()`. This results in an application crash the next time the application is executed in the Visual Basic environment.

  The workaround to this problem is to programmatically set the OrbixCOMet configuration setting `COMET_SHUTDOWN_POLICY` to `"atexit"`.
- Certain versions of regsvr32 have been known to crash when registering a handler DLL. If this occurs, you should use the OrbixCOMet `oleregit.exe` tool located in the *<COMET ROOT>*`\bin` directory instead.

  For example, to register `foo.dll`, type:

  ```
  oleregit foo.dll /REGSERVER
  ```

  To unregister `foo.dll`, type:

  ```
  oleregit foo.dll /UNREGSERVER
  ```
- When uninstalling OrbixCOMet, you might need to unregister COMet DLLs from the OLE registry by running the `unregCOmet.bat` batch file located in the `COMet\bin` directory.
- When using bounded sequences from a COM client that has OrbixCOMet loaded in-process, you should `memset` any unused elements in the sequence to '0'. OrbixCOMet will attempt to skip these unused elements, but you

might receive a marshalling error if the element types are complex.

- `aliassrv.exe` does not work on Window 95.

Installing OrbixCOMet over Existing Orbix/ActiveX Integration

OrbixCOMet and the Orbix Desktop static Active/X integration are incompatible and cannot co-exist on the same machine. This is due to the fact that the Automation ProgIDs are the same. (These ProgIDs are specified by the OMG Interworking specification). The installation program will detect, and at your prompting attempt to automatically disable, the Orbix Active\X integration.

Any existing installation of Orbix/ActiveX Integration must be disabled before installing OrbixCOMet. To do this:

1. `CD $(ORBIX_ROOT)\bin`
2. `regsvr32 /u IOLEM23C.DLL`

Note: You can use `oleregit.exe IOLEM23C.DLL /UNREGSERVER` to unregister the original OLE support library.

If, at a later date, you wish to revert to using Orbix/ActiveX, you can deactivate OrbixCOMet using the `unregCOMet.bat` batch file in the OrbixCOMet `\bin` directory, and reactivate the Orbix Desktop bridge as follows:

1. `CD $(ORBIX_ROOT)\bin`
2. `regsvr32 IOLEM23C.DLL`

If `regsvr32` is not available, or it causes problems, you can use `$(ORBIX_ROOT)\bin\oleregit.exe` as follows to re-register the original OLE support library:

```
oleregit.exe IOLEM23C.DLL /REGSERVER
```

Similarly, to reactivate the OrbixCOMet bridge, use the batch file `regCOMet.bat` in the OrbixCOMet `\bin` directory.

Building/Running Demonstrations

Run-time libraries for PowerBuilder are not included with OrbixCOMet. You will need this runtime installed if you wish to run these demonstrations.

Furthermore, in order to build the C++ CORBA servers in *<COMet Install>*`\demo\corbasrv`, a valid installation of Orbix3.0 is required. If you have existing CORBA servers for some of these (for example, `grid`, `idl_demo`, and so on) that are standard Orbix demonstrations shipped on all platforms, you can use those. To build the C++ COM client demonstrations, you need Microsoft Visual C++ 6.0 or compatible C++ compiler.

The makefiles for the CORBA servers will call `putidl` to insert the IDL into the IFR. They will also call `putit` to register the server in the Orbix Implementation Repository.

Note: C++ COM applications should not be compiled with the `/Og` or the `/Ox` switch (which implies the `/Og` switch). Instead you should use `/Oityb1 /Gs` for release builds. Refer to the COM demonstration makefiles in *<COMet Install>*`\demos\com` for more details. (This is due to a bug in the code optimiser in the Visual C++ compiler.)

Standalone Server Support

OrbixCOMet allows developers, via the `DIOrbixServerAPI` interface, to implement CORBA server objects using languages like Visual Basic, PowerScript, and so on.

An example of how to use this to write a CORBA server called `bank` in Visual Basic is shown in the following example that can be found in the *<COMET ROOT>*`\demo\vb6\bankSrv` demonstration:

```
Dim orb As Object
Set orb = CreateObject("CORBA.ORB.2")
Set serverAPI = orb.GetServerAPI
Set orb = Nothing
' bankObj created earlier (not shown) and is our
' implementation object
Call serverAPI.SetObjectImpl("bank", "", bankObj)
Call serverAPI.Activate("bank")
```

The call to Activate in the preceding example calls `impl_is_ready()` internally. This signifies the server's availability to the network. As a result, an Orbix daemon is required on the machine where this application runs. However, this might not always be the case, so there is also support for writing servers that can run without an Orbix daemon. An example of one such server is shown in the following example that can be found in the *<COMET ROOT>*`\demo\vb6\standAlone` demonstration:

```
Dim orb As Object
Set orb = CreateObject("CORBA.ORB.2")
Set serverAPI = orb.GetServerAPI
Set orb = Nothing
' SetObjectImplPersistent(interface, marker, server, object,
                            filename)
Call serverAPI.SetObjectImplPersistent("bank", "", "bank",
                            _bankObj, "c:\temp\bank.ior")
Call serverAPI.ActivatePersistent
```

As well as specifying the interface / marker / server name, the call to `SetObjectImplPersistent` specifies a file to which the IOR for the object should be written. Prospective clients should then call `CORBA::ORB::string_to_object()` on the IOR. (If you are using the CORBA Factory, you can take advantage of the build in support for IORs in the `GetObject` call. Refer to *<COMET ROOT>*`\demo\vb6\standAlone\vbClient` for an example of this). A server written in this manner can be started persistently, without the need for a daemon on the local machine.

Servers that use the `DIOrbixServerAPI` must now dispatch their own Orbix events (that is, call `serverAPI.dispatchEvents` from within (for example) a Visual Basic timer). Such applications must also set the `COMet.Config.AUTO_EVENTS` configuration value to "no". For example:

```
Set orb = CreateObject("CORBA.ORB.2")
orb.SetConfigValue "COMet.Config.AUTO_EVENTS", "no"
Timer1.Enabled = False

// use server API

Set serverAPI = orb.GetServerAPI
Set orb = Nothing
Call serverAPI.SetObjectImpl("bank", "", bankObj)
Call serverAPI.Activate("bank")
```

```
Timer1.Interval = 500
Timer1.Enabled = True
```

The timer function should look something like the following:

```
Private Sub Timer1_Timer()
  Timer1.Enabled = False
  serverAPI.DispatchEvents
  Timer1.Enabled = True
End Sub
```

Failure to follow this approach might result in marshalling errors. (Such behavior was noticeable in OrbixCOMet 1.0 UR2.)

Stand-Alone IFR

The Orbix IFR can now be run as a stand-alone server (that is, in a configuration without a running Orbix daemon). The `COMetIFR.exe` that was in previous releases of OrbixCOMet has now been merged with the Orbix IFR.

When running the IFR as a stand-alone CORBA server, you must tell it which port to listen on. You should do this from a DOS command prompt.

To use:

1. Set the environment variable `IT_SERVER_PORT` to a port number that will be used by `IFR.exe` when it is saving its IOR. (This must be an environment variable.)
2. Set the value for `COMet.TypeMan.TYPEMAN_IFR_IOR_FILENAME` in the configuration file to the name of a file in which you want to store the IOR for the COMetIFR. This setting is used by the IFR utilities and OrbixCOMet utilities to retrieve the IOR.

   For example, this can be in a batch file such as `\bin\startcometifr.bat`:

   ```
   Set IT_SERVER_PORT=2334
   rem  -n   : run as persistent server
   rem  -O   : Output the IOR
   rem  -t 1 : Timeout after 1 sec
   IFR -n -O -t 1
   start IFR.exe -n
   ```

When using the IFR as a stand-alone server, the IFR utilities (`readifr.exe`, `putidl.exe`, and `rmidl.exe`) should all be used with the `-n` switch.

You can also install this Interface Repository as the default Orbix Interface Repository by registering it with the Orbix daemon via the `putit.exe` tool or the Orbix GUI Server Manager. The following is an example of using the `putit` command:

```
c:\> putit IFR c:\iona\COMet_1.0c\bin\COMetIFR.EXE
```

If you have set the `TYPEMAN_IFR_IOR_FILENAME` entry in your registry, you cannot have the IFR auto-launched by the daemon as well. You should use one approach or the other on any single machine.

OrbixNames

This section describes changes in OrbixNames 3.0.

Compatibility with Other IONA Products

OrbixNames 3.0 has been tested with Orbix 3.0 and OrbixWeb versions 3.1 and 3.1.1.

Note: It is not possible for OrbixNames 3.0 and the Java naming service supplied with OrbixWeb to share the same Bindings Repository.

Functionality Removed from OrbixNames 3.0

The following functionality has been removed from OrbixNames 3.0.

Names Library

The Names library, which contained the IDL pseudo-interface types `LNameComponent` and `LName`, has been removed from OrbixNames.

New Features in OrbixNames 3.0

This section describes the new functionality and major changes added in OrbixNames 3.0.

Documentation

The OrbixNames user documentation has been updated for this release. The OrbixNames user documentation is a single volume, called the *OrbixNames Programmer's and Administrator's Guide*.

Orbix Demos

A load balancing demonstration is provided and resides in the OrbixNames `demos` directory of your Orbix 3.0 installation. This is an elementary demonstration, described in the *OrbixNames Programmer's and Administrator's Guide*.

Configuration

As highlighted for the Orbix 3.0 release notes, there has been a significant overhaul of how an Orbix installation is now configured. Each Orbix 3.0 service now has its own configuration file. The OrbixNames configuration variables are now scoped and defined in the file `orbixnames3.cfg`.

The main configuration variable set is described in *OrbixNames Programmer's and Administrator's Guide*. In addition to these variables, it is now possible when using OrbixNames to configure the format of an IOR, with respect to its host address part. The IOR can contain either the IP address or a host name. The `OrbixNames.IT_USE_HOSTNAME_IN_IOR` variable determines this characteristic. The default value is `true`. With this value, a host name appears in an IOR. Setting the value to false causes the IOR to contain an IP address.

Finding Unreachable Naming Contexts

If a naming context exists in the Naming Service but has no associated name that allows it to be retrieved, OrbixNames puts it in a new naming context, called the `lost+found` context. Refer to the *OrbixNames Programmer's and Administrator's Guide* for more details.

Multi-threading

The OrbixNames server is now a multi-threaded application.

Bugs Cleared in OrbixNames 3.0

This section describes the incidents cleared in this release. All incidents are cross platform unless otherwise stated. The incidents are described in terms of **Incident ID**, **PR Number**, and **Synopsis**, as described on page 20.

Incident ID	PR Number	Synopsis
38000	192357	The command `putnewncns` crashes when a name of more than 600 bytes is specified.
28400	162529	The command `putncns` core dumps if incorrect parameters are specified.
24780	160367	The command `del_group`, when used with the `-n` switch, core dumps if the name exists but the group has been deleted.
34560	187359	The operation `resolve()` returns object references that were previously removed from the Naming Service.
26960	163477	A port number 0 appears in the Naming Service IOR when the OrbixNames server is automatically launched.
26180	163401	The operation `CosNaming::NamingContext::OBfactory()` is not described in the documentation.
30940	176192	The `lsns` command, when used with the `-h` switch, hangs the OrbixNames server if the case of the host name is incorrect.
36580	189815	OrbixNames 1.1 does not work with proxified IORs created by Orbix Wonderwall `iortool`.
51475	216644	The marker in the `ObjectKey` in a string format IOR should be in the form module/interface.

Orbix Wonderwall

This section describes changes made in Orbix Wonderwall 3.0.

Licensing

This release of Orbix Wonderwall requires that you license the IIOP proxy with your Orbix 3.0 license key. The installation script attempts to do this. However, if you enter an invalid license key, the proxy will fail at start-up. To enter a new license key, run the following command on UNIX platforms:

```
<Orbix Wonderwall dir>/bin/install_licence
                          <Orbix Wonderwall dir>/iiopproxy <key>
```

The equivalent command on Windows NT is:

```
<Orbix Wonderwall dir>\bin\licence.exe
                  <Orbix Wonderwall dir>\bin\iiopproxy.exe <key>
```

Development Environments

Orbix Wonderwall 3.0 is available for the following environments:

Operating System	Hardware
Solaris 2.x	SPARC

HP-UX 10.20	HP 9000/800. Requires that either the ANSI C++ compiler (aCC) or the HP-UX patch PHSS_16585 (the aCC runtime) is installed before Orbix Wonderwall binaries can be used.
	
	Specifically, the version of the runtime support library required is:
	
	`/usr/lib/libCsup.1:`
	`HP aC++ B3910B A.01.18 Language Support Library`
	
	This can be determined using the command:
	
	`what /usr/lib/libCsup.1`

Windows NT 4	Intel x86

To use the Orbix Wonderwall GUI utilities, you must have either Sun Microsystems' Java Development Kit (JDK) or Java Runtime Environment (JRE) installed, or have installed Orbix 3.0. During Orbix Wonderwall installation you are asked for the location of the JDK. If you do not specify a location, the utilities attempt to use the JRE installed with Orbix 3.0. If you do not specify a JDK or JRE, and you have not installed Orbix 3.0, the GUI utilities will not work.

Compatibility

Orbix Wonderwall is designed to interoperate with any CORBA ORB that implements version 1.0 or 1.1 of the CORBA Internet Inter-ORB Protocol (IIOP).

New Features in Orbix Wonderwall 3.0

This section describes the new functionality and major changes added in OrbixWonderwall 3.0.

Logging Connections

This release includes the ability to turn off logging of connections. To do this, add the following line to the Orbix Wonderwall configuration file:

```
no-log connections
```

Bugs Cleared in Orbix Wonderwall 3.0

This section describes the incidents cleared in this release. All incidents are cross platform unless otherwise stated. The incidents are described in terms of **Incident ID**, **PR Number**, and **Synopsis**, as described on page 20.

Incident ID	PR Number	Synopsis
42700	203791	Orbix Wonderwall fails to handle IORs with multiple component profiles correctly.

Known Problems, Workarounds and Tips

The known problems, workarounds and tips for Orbix Wonderwall 3.0 are as follows:

Fragmented IIOP 1.1 Headers

Fragmented IIOP 1.1 `Request` and `Reply` headers are not yet supported.

Fragmented Replies and HTTP Tunneling

Sending fragmented `Reply` messages from IIOP 1.1 servers over a HTTP-tunneled connection is not yet supported.

Timing Out of Servers with Transformers

If an activated server that requires use of a server transformer times out or is stopped, Orbix Wonderwall attempts to send a transformed message to the server's activation port. This port is associated with the `orbixd` or `orbixdj` process and causes the daemon to fail with an unmarshalling error. This in turn causes the server to be unavailable to the client.

Host Names and Orbix 2.3c

Orbix 2.3c does not use the host name in an IOR, but uses the host name contained in the object key instead. If you intend to use Orbix 2.3c clients to contact Orbix or OrbixWeb servers behind Orbix Wonderwall, with proxified IORs, you must run the Wonderwall and server on the same host, but using different ports. This problem is fixed in Orbix 3.0.

Contacting an Unregistered Server

The OrbixWeb 3.0 activator, `orbixdj`, produces the following stack trace if Orbix Wonderwall tries to bind to a server that is not registered in the Implementation Repository:

```
java.lang.NullPointerException
  at IE.Iona.OrbixWeb.CORBA.ServerRequest.target(ServerRequest.java)
  at IE.Iona.OrbixWeb.Activator.DJAuthenticationFilter.
     inRequestPreMarshal(DJAuthenticationFilter.java)
  at IE.Iona.OrbixWeb.CORBA.ServerRequest.
     inRequestPreMarshal(ServerRequest.java)
  at IE.Iona.OrbixWeb.CORBA.ServerDispatcher.
     dispatchSpecial(ServerDispatcher.java)
  at IE.Iona.OrbixWeb.CORBA.BOA.processRequest(BOA.java)
  at IE.Iona.OrbixWeb.CORBA.BOA.processOneEvent(BOA.java)
  at IE.Iona.OrbixWeb.CORBA.BOA.processEvents(BOA.java)
  at IE.Iona.OrbixWeb.CORBA.EventHandler.run(EventHandler.java)
  at java.lang.Thread.run(Thread.java)
```

This is fixed in OrbixWeb 3.0 patch 2 and later releases of OrbixWeb.

Further Information

For further information about updates to Orbix, including the latest patches, visit the Orbix Update Center at:

http://www.iona.com/online/support/update/index.html